EXPRESS MAIL LABEL NO.: EV268062159US DATE OF DEPOSIT: NOVEMBER 26, 2003

I hereby certify that this paper and fee are being deposited with the United States Postal Service Express Mail Post Office to Addressee service under 37 CFR § 1.10 on the date indicated below and is addressed to the Mail Stop Patent Application, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450

VENESSA M. URENA
NAME OF PERSON MAILING PAPER AND FEE SIGNATURE OF PERSON MAILING PAPER AND FEE

Inventor(s): Erik J. Burckart
David B. Gilgen
Craig A. Lanzen

EFFICIENT CONNECTION POOL VALIDATION

BACKGROUND OF THE INVENTION

Statement of the Technical Field

[0001] The present invention relates to the field of socket based data communications and more particularly to connection pool management.

Description of the Related Art

[0002] Connection pools represent a significant advance in the art of distributed computing. Prior to the implementation of the connection pool, client processes in a client-server configuration established direct connections with back-end server processes on demand. Often, these direct connections take the form of a socket between two ports, each port residing at a particular network address. Opening a connection between a client process and a server process can require the consumption of significant computing resources. Specifically, first a desired socket must be identified in the client process and requested of the server process. A handshaking process can result in consequence of which the connection can be established between the client and server processes.

[0003] It will be recognized by the skilled artisan that in application where multiple, repeated connections will be required, a tremendous amount of computing resources can be consumed merely opening connections when required. Yet, particularly in data intensive applications involving repeated remote connections to back end database servers, so much can be the case. Further, it needn't always be the case that a connection can be established reliably. In this regard, when a socket cannot be established between the processes, the attempt can fail leaving little recourse for the client process. Unfortunately, the failure to attain a connection for use by the client process can range from too few available connections to malfunctioning connections.

[0004] Connection pooling is widely viewed as the appropriate tool to overcome the clear deficiencies of conventional computer process connectivity. Connection pooling is a technique used by applications which open many connections to a finite number of back-end server processes. The reverse proxy represents one typical application of the connection pool. In the prototypical connection pool, connections are pre-established with back-end server processes. These connections can be organized in a data structure such as an array or list in which the connections can remain idle until provisioned either by a host server process for the benefit of an external client, or directly by a client process. In this way, when client processes require the use of a connection with a back-end server process, it will not be necessary to create the connection each time.

[0005] In managing a pool of idle connections, an efficient process consuming little computing overhead can be required to validate that the connections which not been provisioned recently remain valid and useable on demand. In this regard, it is known to

perform placebo transactions using the idle connections to ensure the reliability of the idle connections. Of course, to perform the placebo transaction with respect to any one idle connection necessarily consumes the idle connection such that any attempt to access the idle connection can be blocked. While the liberal use of threads can overcome the blocking action of the placebo transaction, spawning a great many threads for concurrent usage can consume significant computing resources. Thus, in a pool of multiple idle connections, validating each connection using threaded validation processes can produce the same level of computing overhead that otherwise would exist in consequence of invalid idle connections.

SUMMARY OF THE INVENTION

[0006] The present invention addresses the deficiencies of the art in respect to connection pool management and provides a novel and non-obvious method, system and apparatus for highly efficient connection pool management. In a preferred aspect of the invention, a connection pool management system can include a connection pool configured to store one or more idle connections. The system further can include a connection manager programmed for coupling to the connection pool. The connection manager further can be programmed to validate individual ones of the idle connections by issuing a non-blocking input/output (I/O) operation to each of the individual ones of the idle connections.

[0007] The connection pool can include an array configuration. Each element in the array configuration can include both a timestamp data member and also a reference to one of the idle connections. The connection pool also can include a configuration for a last-in first-out (LIFO) ordering of the idle connections. Finally, the connection pool can include a configuration for storing a global timestamp. The global timestamp can indicate a time value when an oldest one of idle connections had been added to the connection pool.

[0008] The present invention also can include a connection pool management method. In the method of the invention, responsive to adding a first one of multiple idle connections to a connection pool, a global timestamp can be recorded to indicate a time value when the first idle connection had been added to the connection pool. Similarly, responsive to adding subsequent idle connections to the connection pool, individual

timestamps can be recorded in the connection pool in association with corresponding idle connections. A timestamp of an oldest one of the idle connections can be compared to the global timestamp to determine whether a timeout condition has arisen. in consequence, when determining that a timeout condition has arisen, at least one of the idle connections can be probed with a non-blocking I/O request in order to validate the idle connections.

[0009] Additional aspects of the invention will be set forth in part in the description which follows, and in part will be obvious from the description, or may be learned by practice of the invention. The aspects of the invention will be realized and attained by means of the elements and combinations particularly pointed out in the appended claims. It is to be understood that both the foregoing general description and the following detailed description are exemplary and explanatory only and are not restrictive of the invention, as claimed.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] The accompanying drawings, which are incorporated in and constitute part of this specification, illustrate embodiments of the invention and together with the description, serve to explain the principles of the invention. The embodiments illustrated herein are presently preferred, it being understood, however, that the invention is not limited to the precise arrangements and instrumentalities shown, wherein:

[0011] Figure 1 is a schematic illustration of a connection manager configured to manage a connection pool in accordance with the present invention;

[0012] Figure 2 is a flow chart illustrating a process for adding a new connection to the connection pool of Figure 1;

[0013] Figure 3 is a flow chart illustrating a process for validating connections in the connection pool of Figure 1; and,

[0014] Figure 4 is a flow chart illustrating a process for provisioning a connection in the connection pool of Figure 1.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0015] The present invention is a method, system and apparatus for managing a connection pool in a highly efficient manner. In accordance with the present invention an array can be configured to store one or more established connections in a LIFO arrangement. Each occupied entry in the array additionally can store a timestamp indicating first when the connection had been added to the array and subsequently when the connection had last been validated. Importantly, only timestamp for the last added connection can be compared to a pre-established time-out for the pool to determine when the connections in the pool ought to be validated.

[0016] During validation, each connection in the array can be queried to determine whether the timestamp of the connection exceeds the pre-configured time-out. If the timestamp of the connection exceeds the time-out, a non-blocking I/O operation can be applied through the connection to determine whether the connection remains valid. If the non-blocking I/O operation succeeds, the timestamp can be updated to reflect the present time and the next connection in the array can be tested. Otherwise, the connection can be removed from the array. Importantly, the skilled artisan will recognize that by applying merely a non-blocking I/O operation to the connection, rather than spawning an independent thread to test the validity of the connection, substantial consumption of computing resources can be avoided.

[0017] Figure 1 is a schematic illustration of a connection manager configured to manage a connection pool in accordance with the present invention. A connection manager 130 can be coupled to a connection pool 110 and one or more client

processes 140. The client processes 140 can be directly coupled to the connection manager 130 in consequence of which individual ones of the client processes 140 can directly request the use of a connection 120A, 120B, 120n disposed within the connection pool 110. Alternatively, the client processes 140 can request the usage of a connection 120A, 120B, 120n intermediately through a server process (not shown) coupled to the connection manager 130.

[0018] In either case, the connection pool 110 can include one or more idle connections 120A, 120B, 120n. The connections 120A, 120B, 120n can be arranged within the connection pool 110 in an array type data structure. The array type data structure can be accessed by the connection manager 130 in a LIFO manner. As a result, the array type data structure can take the form of a stack in which the most recently added idle connections in the connection pool 110 can be used for subsequent requests to provision a connection. Accordingly, the likelihood of provisioning a valid connection can be enhanced substantially.

[0019] Each of the connections 120A, 120B, 120n in the connection pool 110 can have associated therewith a timestamp. The timestamp can indicate when each individual one of the connections 120A, 120B, 120n had been added to the connection pool 110. The timestamp can be updated at a subsequent time when the individual ones of the connections 120A, 120B, 120n are validated in accordance with the present invention. In any case, the timestamps of the connections 120A, 120B, 120n can be compared with a global timestamp associated with the connection pool 110 to determine when validation will be required. The global timestamp can hold the value of the oldest timestamp in the connection pool 110. When a newly idle one of the

connections 120A, 120B, 120n is added to the connection pool 110, the timestamp of the newly idle one of the connections 120A, 120B, 120n can be compared to the sum of the global timestamp and a pre-configured time-out value. If the timestamp of the newly idle one of the connections 120A, 120B, 120n exceeds the sum, the entire connection pool 110 can be validated.

[0020] During the validation process, the connection manager 130 can synchronize the array so that new additions and removals from the connection pool 110 can be suspended temporarily. Subsequently, the array can be traversed sequentially. At each element in the array, the timestamp of the corresponding one of the connections 120A, 120B, 120n in the array element can be compared to the sum of the global timestamp and the time-out value to determine whether the corresponding one of the connections 120A, 120B, 120n ought to be validated.

[0021] If so, a non-blocking I/O operation 150 can be applied to the corresponding one of the connections 120A, 120B, 120n. If the operation fails, the corresponding one of the connections 120A, 120B, 120n can be invalidated and removed from the connection pool 110. Otherwise, the timestamp can be updated with the latest time and the next one of the connections 120A, 120B, 120n in the connection pool 110 can be processed until no connections 120A, 120B, 120n remain to be validated in the connection pool 110.

[0022] In more particular illustration of the operation of the connection manager 130, Figures 2 through 4 are flow charts illustrating processes for adding, validating and provisioning the connections 120A, 120B, 120n of the connection pool 110 of Figure 1.

Beginning first with Figure 2, a flow chart is shown which illustrates a process for adding a new connection to the connection pool. Beginning first in block 210, a request can be received to pool a connection. In decision block 220, it can be determined whether the pool includes any idle connections, or whether the pool is empty. If the pool is empty, in block 260 the global timestamp of the pool can be updated to reflect the current time and the current time also can be saved in association with the new connection in block 270.

[0023] If in decision block 220, the connection pool is determined to include at least one idle connection, in decision block 230 it can be determined whether the global timestamp exceeds that of the sum of the global time stamp and a pre-established timeout value. If not, the connection can be added to the pool along with a current timestamp in block 240 and the process can end in block 250. Otherwise, in block 280, it can be determined that the entire pool must be considered for pool validation and the process can continue in Figure 3. Notably, when referring to a timestamp herein, it is not intended to infer that all timestamps must indicate a specific time based upon a twenty-four hour clock. Rather, the timestamp equally could include an elapsed time.

[0024] Figure 3 is a flow chart illustrating a process for validating connections in the connection pool of Figure 1. Beginning in block 305, a request to validate the connection pool can be received. In block 310, a first connection from the pool can be retrieved for validation. In decision block 315, it can be determined whether the timestamp of the first connection exceeds that of the global timestamp in combination with the pre-established timeout value. If not, the connection need not be validated further and in decision block 320, if more connections remain to be processed, the next

connection can be extracted from the pool in block 310 and the validation process can continue. Otherwise the process can end in block 360.

[0025] If, in decision block 315, it is determined that the timestamp of the first connection exceeds that of the global timestamp in combination with the pre-established timeout value, in block 325 a non-blocking I/O operation can be performed over the connection. Non-blocking I/O operations are known in the art and complete libraries have been developed for this purpose. In the case where the connection in the connection pool couples a back-end data server to a client process, a non-blocking read can be performed over the connection.

[0026] In any case, in decision block 330, if the non-blocking I/O operation fails, in block 335 the connection can be presumed invalid and closed. In block 340 the connection can be removed from the connection pool and in block 345 the array element can be "plugged" by moving the most recently added idle connection in the pool to the array element vacated by the removed idle connection. In contrast, if the non-block I/O operation does not fail, the connection can be presumed valid and its respective timestamp can be updated with the current time in block 350. In both cases, the process can continue through decision block 320 as described before. Once all of the connections have been validated, the process can end and the idle connections in the pool can be freed for provisioning by the connection manager.

[0027] In this regard, Figure 4 is a flow chart illustrating a process for provisioning a connection in the connection pool of Figure 1. Beginning first in block 405, a connection can be requested and in decision block 410 it can be determined whether any

connections remain available for requisition. If not, in block 440 the connection manager can wait until an idle connection becomes available. Otherwise, in block 415 the last idle connection added to the connection pool can be extracted and in block 420 a non-blocking I/O operation can be performed upon the extracted connection to ensure its validity.

[0028] If in decision block 425 the non-blocking I/O operation does not fail, in block 430 the extracted connection can be returned to the connection manager for use by the requesting process and the process can end in block 435. By comparison, if in decision block 425 the non-blocking I/O operation fails, in block 445 the connection can be determined to be invalid and closed. In block 450, subsequently, the connection can be removed from the pool and the array element holding a reference to the connection can be "plugged" as described before.

[0029] The present invention can be realized in hardware, software, or a combination of hardware and software. An implementation of the method and system of the present invention can be realized in a centralized fashion in one computer system, or in a distributed fashion where different elements are spread across several interconnected computer systems. Any kind of computer system, or other apparatus adapted for carrying out the methods described herein, is suited to perform the functions described herein.

[0030] A typical combination of hardware and software could be a general purpose computer system with a computer program that, when being loaded and executed, controls the computer system such that it carries out the methods described herein.

The present invention can also be embedded in a computer program product, which comprises all the features enabling the implementation of the methods described herein, and which, when loaded in a computer system is able to carry out these methods.

[0031] Computer program or application in the present context means any expression, in any language, code or notation, of a set of instructions intended to cause a system having an information processing capability to perform a particular function either directly or after either or both of the following a) conversion to another language, code or notation; b) reproduction in a different material form. Significantly, this invention can be embodied in other specific forms without departing from the spirit or essential attributes thereof, and accordingly, reference should be had to the following claims, rather than to the foregoing specification, as indicating the scope of the invention.